A New Parallel Method for Molecular Dynamics Simulation of Macromolecular Systems

STEVE PLIMPTON* and BRUCE HENDRICKSON

Parallel Computational Sciences Department 1421, MS 1111, Sandia National Laboratories, Albuquerque, NM 87185-1111. Phone: (505) 845-7873. E-Mail: sjplimp@cs.sandia.gov

ABSTRACT

Short-range molecular dynamics simulations of molecular systems are commonly parallelized by replicated-data methods, in which each processor stores a copy of all atom positions. This enables computation of bonded 2-, 3-, and 4-body forces within the molecular topology to be partitioned among processors straightforwardly. A drawback to such methods is that the interprocessor communication scales as N (the number of atoms) independent of P (the number of processors). Thus, their parallel efficiency falls off rapidly when large numbers of processors are used. In this article a new parallel method for simulating macromolecular or small-molecule systems is presented, called force-decomposition. Its memory and communication costs scale as $N/\sqrt{P}$, allowing larger problems to be run faster on greater numbers of processors. Like replicated-data techniques, and in contrast to spatial-decomposition approaches, the new method can be simply load balanced and performs well even for irregular simulation geometries. The implementation of the algorithm in a prototypical macromolecular simulation code ParBond is also discussed. On a 1024-processor Intel Paragon, ParBond runs a standard benchmark simulation of solvated myoglobin with a parallel efficiency of 61% and at 40 times the speed of a vectorized version of CHARMM running on a single Cray Y-MP processor.

Introduction

Molecular dynamics (MD) is a widely used computational tool for simulating liquids and solids at an atomistic level.[1] Macromolecular systems used as polymers, proteins, and DNA are particularly interesting to study with MD because the conformational shape of the molecules often determines their functional and catalytic properties. Such systems are also computationally challenging to simulate because (1) in the absence of crystal periodicity, large numbers of atoms must often be included in the model; and (2) interesting events, such as molecular diffusion or conformational changes, typically occur on long time scales relative to the femtosecond-scale timesteps of the MD model.

* Author to whom all correspondence should be addressed.

Journal of Computational Chemistry, Vol. 17, No. 3, 326–337 (1996)

CCC 0192-8651 / 96 / 030326-12

MD simulations are natural candidates for implementation on parallel computers because the forces on each atom or molecule can be computed independently.[2-4] In this article we focus on MD simulations of molecular systems which require computation of bonded forces within the topology of the simulated molecules in addition to the standard nonbonded van der Waals and Coulombic forces. To achieve good parallel performance, these bonded computations must be distributed evenly across the processors in conjunction with parellization of the nonbonded pairwise computations. We also limit our scope to short-range MD models in which the nonbonded forces are truncated, so that each atom interacts only with other atoms within a specified cutoff distance. Examples of widely used commercial and research codes which implement this model include CHARMM, GROMOS, AMBER, and DISCOVER. The computation in this case scales linearly with N, the number of atoms, since each atom interacts with a roughly constant number of neighbors. While more accurate, MD models with long-range forces are more expensive to compute with, even if hierarchical methods[5] or multipole approximations[6] are used. The computation in these methods is typically partitioned into short- and long-range portions. The short-range portion involves a direct summation of pairwise interactions with near neighbors and thus can be parallelized by the methods described in this article. The long-range portion requires different strategies to efficiently parallelize[7,8] and is beyond the scope of this article.

The most commonly used technique for parellelizing short-range MD simulations of molecular systems is known as the replicated data (RD) method.[4] Numerous parallel algorithms and simulations have been developed based on this approach.[9-16] Typically, each processor stores a copy of all the atom positions in the simulation. It uses this vector of information to compute nonbonded forces for the subset of atoms assigned to it. The bonded force computation can be simply parallelized in this scheme, since each processor can compute the force between any group of bonded atoms. The drawback to the RD method is that its memory and communication cost scale as N independent of P, the number of processors used. Thus, on large numbers of processors communication costs dominate and the algorithm becomes inefficient.

A competing parallel method is known as geometric or spatial decomposition (SD).[2,3] In this approach the simulation domain is broken into P pieces, one per processor. Each processor computes forces on only the atoms in its subdomain. In the large N limit, the technique scales optimally as N/P, since each processor need only acquire information from processors who own neighboring subdomains. However, SD algorithms have not been as widely used for molecular simulation[8,17-20] due to their complexity (particularly for bonded force computation) and the difficulty of achieving good load balance across processors for irregularly shaped or dynamically changing domains.

We have developed a new parallel algorithm, called force decomposition (FD), which is particularly appropriate for simulations of molecular systems, be they linear chain polymers or large macromolecules. Although nearly as simple to implement as the RD technique, it reduces the communication cost and memory requirements by a factor of $\sqrt{P}$. This allows many more processors to be used efficiently in a given simulation. Although the new algorithm's scaling is not the optimal N/P of the SD methods, it is similar to RD in that it is geometry independent and thus can be simpler to load balance effectively than SD methods. Later in this article we show that when used on more than a few dozen processors, the FD method can offer faster parallel performance than RD techniques, and we argue that it will often be the fastest of the three parallel methods (RD, FD, DS) for many kinds of molecular simulations of moderate size (up to many tens of thousands of atoms).

In earlier work we described FD techniques for simulations with pairwise Lennard-Jones[3] and embedded atom method[21] potentials for metals. The addition of many-body bonded forces is an important complication requiring special treatment; our previous efforts in this direction are briefly discussed in refs. 22 and 23. In this article we detail an enhanced version of the FD method with reduced communication cost (motivated by an algorithm for parallel matrix–vector multiplication discussed in ref. 24). We also provide, for the first time, a full description of how to parallelize the computation of bonded forces within the context of the FD method.

In the next section we outline the computations performed in MD simulations of molecular systems. Then we briefly describe RD and SD methods, to put the new algorithm in context, and detail the FD method. We have implemented both RD and FD algorithms in a macromolecular MD mode called ParBond. We use it to compare the

performance of the two approaches for benchmark simulations of myoblogin and liquid crystal systems.

MD Simulations of Molecules

In MD simulations of molecular systems, two kinds of interactions contribute to the total energy of the ensemble of atoms: nonbonded and bonded. These energies are expressed as simple empirical relations[25]; the desired physics or chemistry is simulated by specifying appropriate coefficients. The energy E_{nb} due to nonbonded interactions is typically written as

$$E_{nb} = \sum_i \sum_j \frac{q_i q_j}{r} + \sum_i \sum_j \epsilon_{ij}\left[\left(\frac{\sigma_{ij}}{r}\right)^{12} - \left(\frac{\sigma_{ij}}{r}\right)^{6}\right] \quad (1)$$

where the first term is Coulombic interactions and the second is van der Waals, r is the distance between atoms i and j, and all subscripted quantities are user-specified constants. In short-range simulations, the summations over i and j are evaluated at each timestep to only include atom pairs within a cutoff distance r_c, such that $r < r_c$. The bonded energy E_b for the system in the harmonic approximation can be written as

$$\begin{aligned} E_b = &\sum_{\text{bonds}} K_b(r - r_0)^2 + \sum_{\text{angles}} K_\theta(\theta - \theta_0)^2 \\ &+ \sum_{\text{dihedrals}} K_{\phi_p}\left[1 + d_p \cos(n_p \phi)\right] \\ &+ \sum_{\text{impropers}} K_\phi(\phi - \phi_0)^2 \end{aligned} \quad (2)$$

where the first term is 2-body energy, the second is 3-body energy, and the last two are 4-body interactions for torsional dihedral and improper dihedral energies within the topology of the molecules. The distance r and angles θ and ϕ are computed for each interaction as a function of the atomic positions; the subscripted quantities are constants. In contrast to the nonbonded energy, the summations in this equation are explicitly enumerated by the user to set up the simulation (i.e., the connectivities of the molecules are fixed). In the MD simulation, derivatives of eqs. (1) and (2) yield force equations for each atom which are integrated over time to generate the motion of the ensemble of atoms.

Replicated Data Algorithm

The first of the two basic methods that have been used to parallellize the computation of eqs. (1) and (2) is replicated data (RD) or atom decomposition algorithms. Each processor is assigned a subset of N/P atoms and updates their positions and velocities for the duration of the simulation, regardless of where they move in the physical domain. In this setting, the computational work involved in evaluating eq. (1) can be represented by the $N \times N$ force matrix F. The (ij) element of F represents the nonbonded force on atom i due to atom j. Note that F is sparse due to short-range forces. To take advantage of Newton's third law, we also set $F_{ij} = 0$ when $i > j$ and $i + j$ is even, and likewise set $F_{ij} = 0$ when $i < j$ and $i + j$ is odd. Conceptually, F is now colored like a checkboard with red squares above the diagonal set to zero and black squares below the diagonal also set to zero. Zeroing half the matrix elements can also be accomplished by striping F in various ways.[26] We also define x and f as vectors of length N which store the position and total force on each atom. For a 3D simulation, x_i would store the three coordinates of atom i.

With these definitions, RD algorithms assign each processor a subblock of F which consists of N/P rows of the matrix, as shown in Figure 1. If z indexes the processors from 0 to $P - 1$, then processor P_z computes nonbonded forces in the F_z subblock of rows. It also is assigned the corresponding subvectors of length N/P denoted by x_z and f_z.

The computation of the nonbonded force F_{ij} requires only the two atom positions x_i and x_j. But to compute all the forces in F_z, processor P_z will need the positions of many atoms owned by other processors. In Figure 1 this is represented by having the horizontal vector x at the top of the figure span all the columns of F. This implies that each processor must store a copy of all N atom positions. To take advantage of Newton's third law, each processor will also compute forces on atoms it does not own, so it must also store a copy of all N forces represented by the force vector f.

Using these definitions, a typical RD algorithm is outlined in Figure 2, with the dominating term in the computation or communication cost of each step listed on the right. We assume at the beginning of the timestep that each processor knows the current positions of all N atoms (i.e., each has an

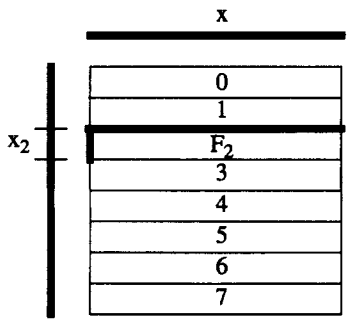

FIGURE 1. The division of the force matrix among eight processors in a replicated data algorithm. Processor 2 is assigned N/P rows of the matrix and the corresponding x_2 piece of the position vector. In addition, it must know the entire position vector x (shown spanning the columns) to compute the nonbonded forces in F_2.

updated copy of the entire x vector. In step (1) of the algorithm, the nonbonded forces in matrix subblock F_z are computed. This is typically done using neighbor lists to tag the interactions that are likely to be nonzero at a given timestep. In the parallel algorithm, each processor constructs lists for its subblock F_z once every few timesteps. As each pairwise nonbonded interaction between atoms i and j is computed, the force components are summed twice into the processor's local copy of f, once in location i and once (negated) in location j, so that F_z is never actually stored as a matrix. Step (1) scales as N/P, the number of nonzero, nonbonded interactions computed by each processor. In step (2) the bonded forces in Eq. (2) are computed. This can be done by spreading the loops implicit in the summations of eq. (2) evenly across the processors. Since each processor knows the positions of all atoms, it can compute any of the terms in eq. (2) and sum the resulting forces into its local copy of f. This step also scales as N/P, since there are a small number of bonded interactions per atom.

(1)	**Compute** non-bonded forces in F_z, doubly summing results into local copy of f	$\frac{N}{P}$
(2)	**Compute** $1/P$ fraction of bonded forces, summing results into local copy of f	$\frac{N}{P}$
(3)	**Fold** f across all processors, result is f_z	N
(4)	**Update** atom positions in x_z using f_z	$\frac{N}{P}$
(5)	**Expand** x_z among all processors, result is x	N

FIGURE 2. Single timestep of the replicated data algorithm for processor P_z.

In step (3) the force vector copies are summed across all P processors in such a way that each processor ends up with the total force on only its N/P atoms. This is called a fold operation[3,27,28] and scales optimally as N, the volume of data in the force vector f. We note that the fold operation is less costly than a global sum operation, in which each processor ends up with the total force on all N atoms, as is done in the RD algorithms discussed in refs. 11, 13, 14, and 16. A global sum operation typically scales as $N \log_2(P)$ and so on 256 processors is eight times more expensive than a fold. The N/P forces resulting from the fold are used to update atom positions and velocities in step (4). Finally, in step (5) the new atom positions in x_z are shared among all P processors in preparation for the next timestep. This is called an expand operation[3,27,28] and is essentially the inverse of the fold operation. Now each processor starts with a small N/P piece of the position vector and ends up with a copy of the entire N-length vector. The cost of this communication step also scales as N.

The chief advantage of the RD algorithm we have outlined is simplicity. It straightforwardly divides the MD force computation and integration evenly across the processors as long as each F_z block has roughly the same number of nonzero elements. This may not be the case if, for example, some atoms have fewer neighbors than others due to nonuniform densities; load imbalance is the result. This can often be overcome by randomly permuting the order of the atoms before the simulation begins. This is a static load balancing method which scatters nonzero elements uniformly throughout the F matrix. Alternatively, Young and Brooks[29] have proposed a dynamic method for preserving load balance in RD algorithms. They adjust the size of the subblocks in Figure 1 as the simulation progresses. Individual processors lose or gain atoms to ensure continual load balance.

The chief drawback of the RD algorithm is that it requires global communication in the fold and expand steps, as each processor must acquire information held by all the other processors. Because this communication scales as N, independent of P, it limits the number of processors that can be used effectively. Similarly, the RD algorithm requires that each processor store a copy of the entire N-length position and force vectors. This memory overhead often limits the size of problems that can be simulated.

Spatial Decomposition Algorithm

The second parallel method for computing eqs. (1) and (2) exploits the locality of the short-range forces by assigning to each of the P processors a small region of the simulation domain, call it D_z, where again z indexes the processors from 0 to $P-1$. This is a geometric or spatial decomposition (SD) of the workload; an algorithm of this form is outlined in Figure 3. Each processor will update the positions x_z of only the atoms in its subdomain. To do this it will need to know not only x_z but also positions y_z of atoms owned by processors whose subdomains are within a cutoff distance r_c of its subdomain. Similarly, as it computes forces f_z on its atoms, it will compute components of forces g_z on the nearby atoms (Newton's third law). With these definitions, steps (1) and (2) of the algorithm are the computation of nonbonded and bonded forces for interactions involving the processor's atoms. These steps scale as the number of atoms N/P in each processor's subdomain. In step (3) the g_z forces computed on neighboring atoms are communicated to processors owning neighboring subdomains. The received forces are summed with the previously computed f_z to create the total force on a processor's atoms. The scaling of this step depends on the length of the force cutoff relative to the subdomain size. We list it as Δ and discuss it further later. Step (4) updates the positions of the processor's atoms. In step (5) these positions are communicated to processors owning neighboring subdomains so that all processors can update their y_z list of nearby atoms. Finally, periodically (usually when neighbor lists are created), atoms which have left a processor's subdomain must be moved in step (6) to the appropriate new processor.

(1) **Compute** non-bonded forces in D_z, summing results into f_z and g_z	$\frac{N}{P}$
(2) **Compute** bonded forces in D_z, summing results into f_z and g_z	$\frac{N}{P}$
(3) **Share** g_z with neighboring processors, summing received forces into my f_z	Δ
(4) **Update** atom positions in x_z using f_z	$\frac{N}{P}$
(5) **Share** x_z with neighboring processors, using received positions to update y_z	Δ
(6) **Move** atoms to new processors as necessary	Δ

FIGURE 3. Single timestep of the spatial decomposition algorithm for processor P_z.

The preceding description ignores many details of an effective SD algorithm[3] but shows that the algorithm's overall scaling is the optimal N/P, as long as the communication costs Δ can be minimized. In the limit of large N/P ratios, Δ scales as the surface-to-volume ratio $(N/P)^{(2/3)}$ of each processor's subdomain. If each processor's subdomain is roughly equal in size to the force cutoff distance, then Δ scales as N/P and each processor receives N/P atom positions from each of its neighboring 26 processors (in 3D). In practice, however, there are several obstacles to minimizing Δ and achieving high parallel efficiencies in MD simulations of molecular systems:

1. Molecular systems may simulated in a vacuum or with surrounding solvent that does not uniformly fill a 3D box. In this case it is nontrivial to divide the simulation domain so that every processor's subdomain has an equal number of atoms in it. Load imbalance is the result.
2. Because of the $1/r$ dependence of Coulombic energies in eq. (1), long cutoffs are often used in simulations or organic materials. Thus a processor's subdomain may be much smaller than the cutoff. The result is considerable extra communication in steps (3) and (5) to acquire needed atom positions and forces (i.e., Δ no longer scales as N/P but as the cube of the cutoff distance r_c).
3. As atoms move to new processors in step (6), molecular connectivity information must be exchanged and updated between processors. The extra coding to manipulate the appropriate data structures and optimize the communication performance of the data exchange subtracts from the parallel efficiency of the algorithm.

We discuss several SD implementations[17,19,20] later in this article.

Force Decomposition Algorithm

We now present the new force decomposition (FD) algorithm. Its communication cost lies in between that of the RD and SD approaches. The FD method partitions the force matrix F by blocks rather than by rows. Block decompositions of ma-

trices are common in linear algebra algorithms for parallel machines,[24,30,31] which sparked our interest in the idea for short-range MD simulations. The assignment of subblocks of the force matrix to processors with a calendar ordering of the processors is depicted in Figure 4. We assume for ease of exposition that P is an even power of 2, although it is straightforward to implement the algorithm on any number of processors which can be logically mapped to a square or rectangular grid. As before, we let z index the processors from 0 to $P-1$; processor P_z owns and will update the N/P atoms stored in the subvector x_z.

The block decomposition in Figure 4 is actually of a permuted force matrix F' which is formed by rearranging the columns of the original checkerboarded F in a particular way. If we order the x_z pieces in row order (across the rows of the matrix), they form the usual position vector x, which is shown as a vertical bar at the left of the figure. Were we to have x span the columns, as in Figure 1, we would form the force matrix as before. Instead, we span the columns with a permuted position vector x', shown as a horizontal bar at the top of Figure 4, in which the x_z pieces are stored in column order (down the columns of the matrix). Thus, in the 16-processor example shown in the figure, x stores each processor's piece in the usual order (0, 1, 2, 3, 4, ..., 14, 15) while x' stores them as (0, 4, 8, 12, 1, 5, 9, 13, 2, 6, 10, 14, 3, 7, 11, 15). Now the (ij) element of F' is the force on atom i in vector x due to atom j in permuted vector x'.

The F'_z subblock owned by each processor P_z is of size $(N/\sqrt{P}) \times (N/\sqrt{P})$. To compute the nonbonded forces in F'_z, processor P_z must know one $N/\sqrt{P}$-length piece of each of the x and x' vectors, which we denote as x_α and x'_β. As these elements are computed, they will be accumulated into corresponding force subvectors f_α and f'_β. The Greek subscripts α and β each run from 0 to $\sqrt{P}-1$ and reference the row and column position occupied by processor P_z. Thus for processor 6 in the figure, x_α consists of the x subvectors (4, 5, 6, 7) and x'_β consists of the x' subvectors (2, 6, 10, 14).

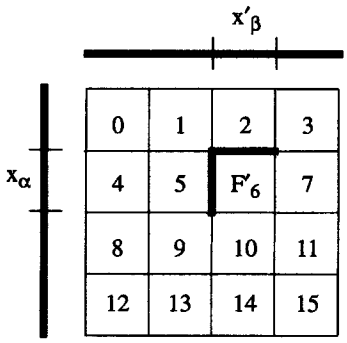

FIGURE 4. The division of the permuted force matrix F' among 16 processors in the force decomposition algorithm. Processor P_6 is assigned a subblock F'_6 of size $N/\sqrt{P}$ by $N/\sqrt{P}$. To compute the nonbonded forces in F'_6, it must know the corresponding $N/\sqrt{P}$-length pieces x_α and x'_β of the position vector x and permuted position vector x'.

The FD algorithm is outlined in Figure 5. As before, each processor has updated copies of the needed atom positions x_α and x'_β at the beginning of the timestep. In step (1), the nonbonded forces in matrix subblock F'_z are computed. As before, neighbor lists can be used to tag the $O(N/P)$ nonzero interactions in F'_z. As each force is computed, the result is summed into the appropriate locations of both f_α and f'_β to account for Newton's third law. In step (2) each processor computes an N/P fraction of the bonded interactions. Since each processor knows only a subset of atom positions, this must be done differently than in the RD algorithm. We describe this computation in the next section.

In step (3) the force on each processor's atoms is acquired. The total force on atom i is the sum of elements in row i of the force matrix minus the sum of elements in column i', where i' is the permuted position of column i. Step (3a) performs a fold within each row of processors to sum the first of these contributions. There is a key difference between this fold operation and the replicated data (RD) fold of Figure 2. In this case the vector f_α being folded is only of length $N/\sqrt{P}$ and only the $\sqrt{P}$ processors in one row are participating in the fold. Thus this operation scales as $N/\sqrt{P}$ instead of N as in the RD algorithm. Simi-

(1) **Compute** non-bonded forces in F'_z, storing results in f_α and f'_β	$\frac{N}{P}$
(2) **Compute** $1/P$ fraction of bonded forces, storing results in f_α and f'_β	$\frac{N}{P}$
(3a) **Fold** f_α within row α, result is f_z	$\frac{N}{\sqrt{P}}$
(3b) **Fold** f'_β within column β, result is f'_z	$\frac{N}{\sqrt{P}}$
(3c) **Subtract** f'_z from f_z, result is total f_z	$\frac{N}{P}$
(4) **Update** atom positions in x_z using f_z	$\frac{N}{P}$
(5a) **Expand** x_z within row α, result is x_α	$\frac{N}{\sqrt{P}}$
(5b) **Expand** x_z within column β, result is x'_β	$\frac{N}{\sqrt{P}}$

FIGURE 5. Single timestep of the force decomposition algorithm for processor P_z.

larly, in step (3b) a fold is done within each column of F'. The two contributions to the total force are joined in step (3c). In step (4), f_z is used to update the N/P atom positions in x_z. Steps (5a–5b) share these updated positions with all the processors that will need them for the next timestep. These are the processors which share a row or column with P_z. First, in (5a), the processors in row α perform an expand of their x_z subvectors so that each acquires the entire x_α. As with the fold, this operation scales as the $N/\sqrt{P}$ length of x_α instead of as N, as it did in the RD algorithm. Similarly, in step (5b), the processors in each column β perform an expand of their x_z. As a result, they all acquire x'_β and are ready to begin the next timestep.

In the FD method, processors will have equal work to do only if all the matrix blocks F'_z are uniformly sparse. If the atoms are ordered geometrically, this will not be the case even for problems with uniform density. This is because such an ordering creates a force matrix with diagonal bands of nonzero elements. As in the RD case, a random permutation of the atom ordering produces the desired effect.

The key feature of the FD method is that the communication operations in steps (3) and (5) now scale as $N/\sqrt{P}$ rather than as N, as was the case with the RD algorithm. Likewise, memory costs for position and force vectors are reduced by the same $\sqrt{P}$ factor. When run on large numbers of processors, this can significantly reduce the time spent in communication. It is important to note that the communication cost of the algorithm is independent of the force cutoff distance, unlike SD methods, whose communication cost grows as the cube (volume) of the cutoff. The FD method also retains the simplicity of the RD technique; it can be implemented using the same expand and fold communication routines.

Bonded Forces

We now return to step (2) of the FD algorithm of the preceding section—computation of bonded forces in the MD model. Because each processor only stores two $N/\sqrt{P}$-length portions of the atom positions, we must ensure that some processor knows the 2, 3, or 4 atom positions required to compute every one of the bonded interactions listed in eq. (2). Similar to the randomization process for load balancing purposes, this can be accomplished as a preprocessing step by a proper ordering of the atoms. We wish to assign atoms to processors in such a way that all bonded interactions can be computed while preserving load balance.

We define a *cluster* of atoms as a collection of one or more atoms. They are not necessarily connected within the topology of a molecule. We also define a *group* of processors as all the processors in one row of the force matrix F' of Figure 5. Thus there are $\sqrt{P}$ processors in each of $\sqrt{P}$ groups. Our preprocessing algorithm proceeds in three stages, each of which is computationally simple and can be performed quickly on a workstation, even for very large MD simulations:

(Stage 1) Create clusters.
(Stage 2) Assign clusters to groups.
(Stage 3) Assign atoms to processors within each group.

The first stage is begun by assigning each atom to its own cluster. We then process the list of 4-body interactions in a random order. We demand that each set of four interacting atoms be in no more than two clusters. If this is not the case, we merge the two smallest clusters to form a new single cluster. This is done twice if the four atoms were initially in four different clusters. When all 4-body interactions have been processed, the list of 3-body interactions is treated similarly. For each set of three atoms, clusters are merged as needed to satisfy the constraint. The goal of this stage is to maximize the total number of clusters, with each cluster being of minimal size. Typically clusters will grow no larger than a dozen or two atoms, and there will be many small one- or two-atom clusters, particularly if small solvent molecules are included in the model.

The second stage of the algorithm is begun by sorting the clusters by size. Beginning with the largest, the clusters are then assigned one by one to whichever group of processors currently has the least number of total atoms. The goal of this stage is to assign equal numbers of atoms to each group; this is simple in practice since there are typically an order of magnitude or more clusters than groups.

In the final stage, the atoms in each group are assigned to individual processors within the group. This is done randomly so that atoms within a cluster end up spread across the $\sqrt{P}$ processors of the group and so that each processor ends up with roughly the same number of total atoms. The only

exception to this rule is as follows. First, the list of 4-body interactions is scanned in a random order once more. If two of the four atoms are in one group and two in another, then both the atoms in one of the sets are assigned to the same (random) processor within their group.

If SHAKE[32] or RATTLE[33] is being used to constrain bond lengths, these stages can be modified to treat small subsets of coupled atoms as one unit. This ensures that all the atoms in a particular subset are assigned to one processor so that computation of the constraint can be parallelized over subsets.

When the stages are completed, each processor will have nearly equal numbers of atoms assigned to it. The randomness used in the stages also ensures that each processor will have roughly equal numbers of nonbonded forces to compute. Moreover, we have guaranteed that every bonded computation will be computable by one or more processors. This is automatic for the 2-body interactions, since some processor will own atom i in its x_α and atom j in its x'_β. For 3-body interactions, at least two of the three atoms will be in a particular x_α, since they were assigned to the same cluster. Some processor in the α group will own the third atom in its x'_β. A similar argument holds for the 4-body interactions. The purpose of the final-stage restriction on two of the four atoms being assigned to the same processor (not just the same group) was to ensure that both atoms will be in the same x'_β.

Once atoms are assigned to processors, the final preprocessing step is to create lists of bonded interactions that will be computed by each processor in the MD simulation. When all the two, three, or four atoms of an interaction are in one x_α (or x'_β), any processor in that row (or column) of F' can compute the interaction. These can be assigned so as to tune the overall load balance of the bonded computations.

Results

We have implemented both RD and FD algorithms in a parallel MD code we have written called ParBond. It is similar in concept (though not in scope) to the widely used commercial and academic macromolecule codes CHARMM, AMBER, GROMOS, and DISCOVER. In fact, ParBond was designed to be CHARMM-compatible in the sense that it uses the same force equations as CHARMM.[25] Since the RD and FD methods both use the same communication primitives, ParBond simply has a switch that partitions the force matrix either by rows or subblocks, as in Figures 1 and 4.

Brooks et al.[9] have done extensive benchmarking with CHARMM on a variety of machines with a large prototypical protein simulation. A 2534-atom myoglobin molecule (with an adsorbed CO) is surrounded by a shell of solvent water molecules for a total of 14,026 atoms. The resulting ensemble is roughly spherical in shape. The benchmark is a 1000-timestep simulation performed at a temperature of 300°K with a nonbonded force cutoff of 12.0 Å. Neighbor lists are created every 25 timesteps with a 14.0-Å cutoff; a total of 6.7 million atom pairs are stored in the neighbor lists. Timing results for the benchmark simulation run on different numbers of processors are shown in Figure 6.

All of the solid symbols in the figure are timings due to Brooks et al.[9] The single processor Cray Y-MP timing of 3.64 s/timestep is for a version of CHARMM they have optimized for vector processing. They have also developed a parallel version of CHARMM[9] using an RD algorithm similar to that discussed earlier in this article. Timings with that version on an Intel iPSC/860 and the Intel Delta at CalTech are shown in Figure 6, as are timings for ParBond using the RD algorithm, running on the Intel Paragon at Sandia. Taking into account that the i860XP floating point processors in the Paragon are about 30% faster than the i860XR chips in the iPSC/860 and Delta and that interprocessor communication is signifi-

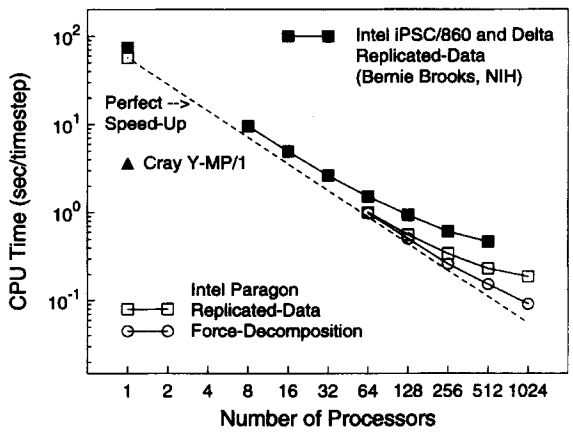

FIGURE 6. Central processing unit (CPU) timings (seconds / timestep) for the replicated data and force decomposition algorithms on different numbers of processors for a 14,026-atom myoglobin benchmark. Timings for the solid symbols are from ref. 9.

cantly faster on the Paragon, the two sets of RD timings are similar. Both curves show a marked roll-off in parallel efficiency above 64–128 processors due to the poor scaling of the all-to-all communication steps.

The timing results for ParBond using the FD algorithm are shown as open circles in Figure 6. The performance falls of less rapidly as processors are added; it is running 1.3 times faster than its RD counterpart on 256 processors (0.265 s/timestep vs. 0.347) and 2.1 times faster on 1024 processors (0.0913 s/timestep vs. 0.189). The 1024-processor timing is about 40 times faster than the single Y-MP processor timing. The dotted line in Figure 6 represents perfect speedup or 100% parallel efficiency for the ParBond code. The benchmark requires too much memory to run on a single processor, so we estimated the ParBond one-processor timing by summming the 64-processor computation times for bonded and nonbonded forces and neighbor list construction only, not communication or load imbalance times. (The one-processor Intel iPSC/860 timing for CHARMM is also an estimate.[9]) The FD algorithm still has a relatively high parallel efficiency in ParBond of 61% on 1024 processors, as compared to 30% for the RD timing.

We have compared the performance of the new FD algorithm to the results in ref. 9, not only because Brooks et al. have provided a standardized benchmark but because their timings exhibit the best scaling of any RD implementation we have seen in the literature (62% parallel efficiency on 128 Intel iPSC/860 processors). These include a parallelization of CHARMM for a 32-processor Intel iPSC/860,[14] of a CHARMM-like code with long-range forces for a 24-processor transputer machine,[12] of AMBER for a 128-processor nCUBE[11] and 512-processor Fujitsu AP1000,[15] of GROMOS for a 64-processor nCUBE and 128-processor Intel iPSC/860,[10] and of general molecular simulation codes for a 64-processor Intel iPSC/860[16] and IBM workstation cluster.[13] All of these efforts show reduced parallel efficiencies as more processors are used due to the scaling problems inherent in the RD approach. Depending on the parallel machine and its communication characteristics, the reported efficiencies fall as low as 10–15% on a few dozens to hundreds of processors, and in some cases the overall speedup is even reduced as more processors are added. The implementation of Sato et al.[15] is a notable exception, with parallel efficiencies of 32 and 44% on 512 processors for their two benchmark calculations.

A comparison of ParBond performance across parallel machines for the RD and FD algorithms is shown in Table I. These are results from simulations of a $3 \times 3 \times 3$ array of 250-atom liquid crystal molecules (6750 total atoms) done in collaboration with Wright Patterson AFB researchers to study atomistic effects on macroscopic structure in thin-film geometries.[34] The cutoffs for forces and neighbor lists used in this model were 8.0 and 10.0 Å. The performance was tested on three parallel machines: a 1024-processor nCUBE2 and Intel Paragon at Sandia and a 512-processor Cray T3D at Cray Research. As in Figure 6, the difference in total runtime (second number in each entry) between the two algorithms grows larger as the number of processors increases for all three machines. The listed communication times (first number in each entry) are the key reason for the difference in performance.

It is worth reemphasizing that the performance of both the RD and FD algorithms scale linearly with N, the number of atoms. Thus the FD advantage of a 1.3–3.3× speed-up on 256–1024 Paragon processors in Figure 6 and Table I should hold for similar simulations of any size system. Because of

TABLE I.
ParBond CPU timings (seconds per 100 timesteps) for replicated data (RD) and force decomposition (FD) algorithms on varying numbers of processors (P) on three parallel machines for a 6750-atom liquid crystal simulation.

	nCUBE2		Intel Paragon		Cray T3D	
P	RD	FD	RD	FD	RD	FD
64	21.8 / 62.7	5.04 / 48.5	3.05 / 12.1	.884 / 9.51	2.11 / 8.55	.745 / 7.28
128	22.0 / 43.8	4.04 / 27.5	3.16 / 8.30	.754 / 5.52	2.16 / 5.60	.667 / 4.19
256	22.2 / 34.1	2.89 / 15.6	3.55 / 6.96	.681 / 3.32	2.22 / 4.19	.586 / 2.60
512	22.2 / 29.2	2.35 / 9.67	3.68 / 6.13	.623 / 2.24	2.38 / 3.75	.547 / 2.13
1024	22.3 / 26.8	1.76 / 6.41	3.96 / 5.98	.651 / 1.78	—	—

The first number in each entry is for communication; the second is total time.

the lower memory requirements for the FD algorithm, we can model systems with as many as 2 million atoms in the ParBond liquid crystal simulations (8.0 Å cutoff) discussed earlier on 1024 processors of the Paragon (16 Mybtes/processor). By contrast, the largest liquid crystal system we can model on that many processors with the RD option in ParBond is about 125,000 atoms. Similarly, Brooks et al. report that their 14,026-atom myoglobin benchmark (12.0 Å cutoff) is nearly the largest simulation they can perform using an RD technique on the 512-processor Intel Delta (about 12 usable Mbytes/processor) with parallel CHARMM.[9]

Finally, while we have not written a spatial decomposition code suitable for molecular systems, we contrast the performance of all three parallel algorithms in an atomic MD simulation. In this benchmark (described fully in ref. 3) individual atoms interact via only a Lennard-Jones potential, analogous to the nonbonded component of a molecular simulation. Figure 7 shows the timing results for simulations of a 3D box containing 10976 atoms in a liquid state (reduced density $\rho^* = 0.8442$) on the nCUBE2 at Sandia. For a force cutoff of 2.5σ (55 neighbors/atom), the SD algorithm is faster than the other two for more than 32 processors. However, for a force cutoff of 5.0σ (440 neighbors/atom), more typical of the longer-range cutoffs used in molecular systems with Coulombic effects, the FD algorithm is the fastest choice on any number of processors. This is because its communication costs are independent of the cutoff length, whereas the SD algorithm must perform more communication as the cutoff length is increased. The performance of the SD code would be further reduced if atoms were not uniformly distributed within a regular 3D box, as in this benchmark. If N were increased (for fixed P), there would come a crossover point at which the SD code becomes faster[3] even for the longer cutoff problem, but these results indicate that there is a regime (up to potentially many tens of thousands of atoms) in which the FD approach will be the fastest algorithmic choice.

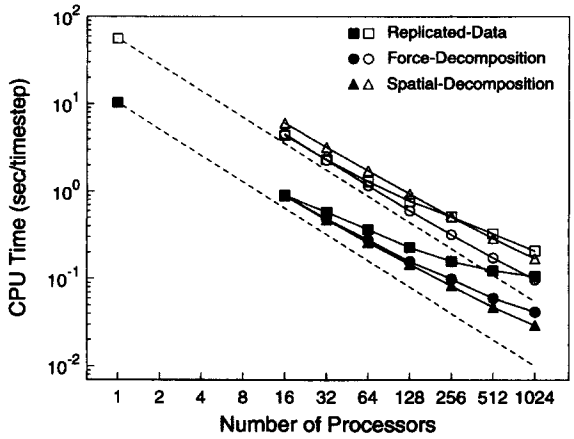

FIGURE 7. CPU timings (seconds / timestep) for replicated data, force decomposition, and spatial decomposition algorithms for a 10,976-atom Lennard-Jones benchmark on different numbers of nCUBE2 processors. The solid symbols are for a 2.5σ cutoff; the open symbols are for a 5.0σ cutoff.

Spatial decomposition implementations of molecular simulations on parallel machines with hundreds of processors are discussed in refs. 17, 19, and 20. Esselink and Hilbers[20] developed their linear-chain polymer model for a 400-processor T800 Transputer machine. They partitioned the simulation box in 2D columns with the third dimension owned wholly on the processor. For a 1400-particle benchmark system with cutoffs that encompassed 200 neighbors/atom, they achieved a 50% parallel efficiency on 400 processors.[20] Brown et al.[17] implemented an SD algorithm with 3D subdomains for linear-chain polymers in rectangular boxes. They reported an efficiency of about 40% on a 64,000-atom simulation on 512 processors of the Fujitsu AP1000. The recently developed EulerGROMOS code of Clark et al.[19] is designed for general biomolecules (with arbitrary connectivity patterns) and simulations of irregular-shaped domains. EulerGROMOS outperforms the earlier parallel replicated data implementation of GROMOS[10] for 128 or more processors of the Intel Delta at CalTech. By using a hierarchical decomposition, each processor in EulerGROMOS ends up with a rectangular-shaped subdomain of variable size which may not align with its neighbors. This allows irregular-shaped domains to be partitioned across processors in a load-balanced fashion at the cost of extra communication overhead. Clark et al. reported a parallel efficiency of roughly 15% (without load balancing) on 512 processors of the Delta for a 10,914-atom simulation of solvated myoglobin with a 10.0-Å cutoff in a uniformly filled 3D box and have simulated systems with up to 131,663 atoms at 25% parallel efficiency.[19]

Conclusions

We have proposed a new strategy, force decomposition, for parallelizing short-range MD simula-

tions and illustrated its effectiveness on several parallel supercomputers. Like replicated data and spatial decomposition algorithms, the FD method's computational cost scales optimally as N/P. Its communication and memory costs scale as $N/\sqrt{P}$, in between the N scaling of RD and N/P scaling of SD.

The new FD algorithm is particularly well suited for simulations of molecular systems in which long force cutoffs, computation of bonded forces, and irregular simulation geometries can degrade the parallel efficiency of SD implementations. Like RD techniques, the FD method is geometry independent, so that even irregular or dynamically changing problems are automatically load balanced. The key advantage of the FD method over RD is its reduced memory costs and improved communication scaling, which enables larger numbers of processors to be used more effectively to simulate a given problem (though in practice the performance difference between RD and FD methods may not become significant until more than a few dozen processors are used). While SD algorithms are clearly the method of choice for very large MD simulations due to their optimal N/P scaling, the relatively high parallel efficiencies and overall simplicity of the FD approach make it a fast option for many of the simulations currently being performed on massively parallel computers.

Acknowledgments

We thank Richard Judson and Grant Heffelfinger of Sandia for fruitful discussions regarding MD simulations of macromolecular systems and for suggesting improvements to a preliminary version of this manuscript. We also thank Grant for helping write the ParBond code and Ron Oldfield of Sandia for coding the bonded-force partitioning algorithm presented in this article. We thank Bernie Brooks of the National Institutes of Health for providing us with input files for the myoglobin benchmark problem. The liquid crystal simulations were performed in collaboration with Wright Patterson AFB researchers Ruth Pachter and Soumya Patnaik. The Cray T3D runs were performed on a machine at Cray Research with the assistance of Barry Bolding. Finally, the bonded-force routines [eq. (2)] in ParBond were written using a public domain MD code authored by Andreas Windemuth (now at Columbia University) as a guide.

This work was partially supported by the Applied Mathematical Sciences program, U.S. Department of Energy, Office of Energy Research, and was performed at Sandia National Laboratories, operated for the DOE under contract no. DE-AC04-76DP00789.

References

1. M. P. Allen and D. J. Tildesley, *Computer Simulation of Liquids*, Clarendon Press, Oxford, 1987.
2. D. Fincham, *Molec. Sim.*, **1**, 1 (1987).
3. S. J. Plimpton, *J. Comp. Phys.*, **117**, 1 (1995).
4. W. Smith, *Comp. Phys. Comm.*, **62**, 229 (1991).
5. J. E. Barnes and P. Hut, *Nature*, **324**, 446 (1986).
6. L. Greengard and V. Rokhlin, *J. Comp. Phys.*, **73**, 325 (1987).
7. H. Q. Ding, N. Karasawa, and W. A. Goddard III, *J. Chem. Phys.*, **97**, 4309 (1992).
8. A. Windemuth, *Parallel Computing in Computational Chemistry*, T. G. Mattson, Ed., American Chemical Society, Washington DC, 1995, p. 151.
9. B. R. Brooks and M. Hodošček, *Chem. Design Automat. News*, **7**, 16 (1992).
10. T. W. Clark, J. A. McCammon, and L. R. Scott, in *Proc. 5th SIAM Conference on Parallel Processing for Scientific Computing*, J. Dongarra, K. Kennedy, P. Messina, D. E. Sorensen, R. G. Voigt, Eds., SIAM, Philadelphia, PA, 1992, p. 338.
11. S. E. DeBolt and P. A. Kollman, *J. Comp. Chem.*, **14**, 312 (1993).
12. H. Heller, H. Grubmuller, and K. Schulten, *Molec. Sim.*, **5**, 133 (1990).
13. J. F. Janak and P. C. Pattnaik, *J. Comp. Chem.*, **13**, 1098 (1992).
14. S. L. Lin, J. Mellor-Crummey, B. M. Pettit, and G. N. Phillips, Jr., *J. Comp. Chem.*, **13**, 1022 (1992).
15. H. Sato, Y. Tanaka, H. Iwama, S. Kawakika, M. Saito, K. Morikami, T. Yao, and S. Tsutsumi, In *Proc. Scalable High Performance Computing Conference-92*, IEEE Computer Society Press, Los Alamitos, CA, 1992, p. 113.
16. W. Smith and T. R. Forester, *Comp. Phys. Comm.*, **79**, 52 (1994).
17. D. Brown, J. H. R. Clarke, M. Okuda, and T. Yamazaki, *Comp. Phys. Comm.* **83**, 1 (1994).
18. S. Chynoweth, U. C. Klomp, and L. E. Scales, *Comp. Phys. Comm.*, **62**, 297 (1991).
19. T. W. Clark, R. V. Hanxleden, J. A. McCammon, and L. R. Scott, In *Proc. Scalable High Performance Computing Conference-94*, IEEE Computer Society Press, Los Alamitos, CA, 1994, p. 95.
20. K. Esselink and P. A. J. Hilbers, *J. Comp. Phys.*, **106**, 108 (1993).
21. S. J. Plimpton and B. A. Hendrickson, In *Materials Theory and Modeling*, J. Broughton, P. Bristowe, J. Newsam, Eds., vol. 291, Materials Research Society Symposium Proc., Materials Research Society, Pittsburgh, PA, 1992, p. 37.
22. B. A. Hendrickson and S. J. Plimpton, *J. Par. and Dist. Comp.*, **27**, 15 (1995).

23. S. J. Plimpton, B. A. Hendrickson, and G. S. Heffelfinger, In *Proc. 6th SIAM Conference on Parallel Processing For Scientific Computing*, R. F. Sincovec, D. E. Keyes, M. R. Leuze, L. R. Petzold, and D. A. Reed, Eds., SIAM, Philadelphia, PA, 1993, p. 178.
24. J. G. Lewis and R. A. van de Geijn, In *Proc. Supercomposing '93*, IEEE Computer Society Press, Los Alamitos, CA, 1993, p. 484.
25. B. R. Brooks, R. E. Bruccoleri, B. D. Olafson, D. J. States, S. Swaminathan, and M. Karplus, *J. Comp. Chem.*, **4**, 187 (1983).
26. H. Schreiber, O. Steinhauser, and P. Schuster, *Parallel Computing*, **18**, 557 (1992).
27. M. Barnett, L. Shuler, S. Gupta, D. Payne, R. van de Geijn, and J. Watts, In *Proc. Supercomputing '94*, IEEE Computer Society Press, Los Alamitos, CA, 1994, p. 107.
28. G. C. Fox, M. A. Johnson, G. A. Lyzenga, S. W. Otto, J. K. Salmon, and D. W. Walker, *Solving Problems on Concurrent Processors: Volume 1*, Prentice Hall, Englewood Cliffs, NJ, 1988.
29. W. S. Young and C. L. Brooks, III, *J. Comp. Chem.*, **16**, 715 (1995).
30. R. H. Bisseling and J. G. G. van de Vorst, In *Lecture Notes in Computer Science, Number 384*, G. A. van Zee and J. G. G. van de Vorst, Eds., Springer-Verlag, 1989, p. 61.
31. B. Hendrickson and D. Womble, *SIAM J. Sci. Stat. Comput.*, **15**, 1201 (1994).
32. J. P. Ryckaert, G. Ciccotti, and H. J. C. Berendsen, *J. Comp. Phys.*, **23**, 327 (1977).
33. H. C. Andersen, *J. Comp. Phys.*, **52**, 24 (1983).
34. S. S. Patnaik, S. J. Plimpton, R. Pachtor, and W. W. Adams, *Liquid Crystals*, **19**, 213 (1995).